UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/583,097	08/02/1999	Marc Tremblay	004-1391-1	7166

62663 7590 11/17/2006

DARBY & DARBY, P.C.
P.O. BOX 5257
NEW YORK, NY 10150-5257

EXAMINER	
HUISMAN, DAVID J	
ART UNIT	**PAPER NUMBER**
2183	

DATE MAILED: 11/17/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 09/583,097
Filing Date: August 02, 1999
Appellant(s): TREMBLAY, MARC

David W. O'Brien, Reg. No. 40,107
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed on June 9, 2005 appealing from the Office action mailed on October 6, 2004.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

In addition, the amendment after final rejection filed on June 9, 2005, has been entered.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

Sites et al., U.S. Patent No. 5,193,167, March, 1993

The Free On-line Dictionary of Computing © 1993-2001 (definitions of "non-deterministic" and "nondeterminism")

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

2. Claims 9-34 are rejected under 35 U.S.C. 102(b) as being anticipated by Sites et al., U.S. Patent No. 5,193,167 (herein referred to as Sites).

3. Referring to claim 9, Sites has taught a superscalar processor that, for a given instruction instance, performs, over plural execution cycles of the superscalar processor, instruction grouping for dispatch, including both intra-group and inter-group dependency checking, wherein the instruction grouping for dispatch takes the plural execution cycles to complete. See Fig.3 and column 6, lines 22-26, and note that dual issue is possible (issuing a group of two instructions), which makes the processor superscalar (able to issue/execute multiple instructions per cycle). In addition, see column 10, lines 38-41, and note that the swap stage of the pipeline (which corresponds to a first cycle) performs intra-group dependency checking as discussed in column 9, lines 45-63. That is, intra-group dependency checking checks the types of the two fetched instructions and based on the types, determines if they may be issued together. This qualifies as dependency checking because the issuing of instruction A is dependent on instruction B's type (where instructions A and B are the two instructions fetched in any given

cycle). Furthermore, see column 10, lines 41-45, and column 6, lines 18-21, and note that after decoding, another dependency check is performed to make sure resources are available. These are inter-group dependency checks because the only reason why resources wouldn't be available for the two instructions is if other instructions are already using those resources (note that this must be the case because resources have not yet been allocated to the fetched instructions). So, this type of dependency check occurs between instructions of different groups. Note also that the intra-group and inter-group checks occur over multiple cycles (stages S0-S3), specifically in stage S1 and again after stage S2 (decode stage) of the pipeline shown in Fig.7. The location of these checks is further illustrated in Fig.3, where it can be seen that the intra-group checks occur at the issue check logic between the instruction register and the decoders and the inter-group checks occur at the issue check logic subsequent to the decoders.

4. Referring to claim 10, Sites has taught a processor as described in claim 9. Sites has further taught:

a) grouping logic implementing plural early pipeline stages of the superscalar processor. See Fig.7 and column 10, lines 64-67, and note that the grouping logic is within the first 4 stages of the pipeline (S0 to S3).

b) plural execution units of varying pipeline depth coupled to receive instructions dispatched from the grouping logic. See Fig.4 and Fig.6 and note the integer and floating-point units. Also, see column 8, line 68, to column 9, line 5, and note that the floating-point divide requires multiple cycles.

5. Referring to claim 11, Sites has taught a processor as described in claim 9. Sites has further taught that the instruction grouping identifies successive groups of instructions from an

instruction stream for dispatch to respective ones of plural execution units of the superscalar processor. It is inherent that the system will continue grouping and issuing multiple instructions per cycle if resources are available. Consequently, previous groups, current groups, and subsequent groups will all be issued.

6. Referring to claim 12, Sites has taught a processor as described in claim 11. Sites has further taught that the superscalar processor dispatches all instructions from a particular one of the successive groups before dispatching any instructions from a subsequent one of the successive groups. See column 6, lines 26-38 and note that a remaining instruction from a group will be issued before an instruction of another group.

7. Referring to claim 13, Sites has taught a processor as described in claim 9. Sites has further taught that the intra-group dependency checking spans at least two of the plural execution cycles. See column 10, lines 35-41. Note that two cycles are required to fetch multiple instructions and check to see if they can be issued together (stages S0 and S1). It should be realized that fetching is considered a portion of intra-group dependency checking because if the instructions aren't fetched, then they cannot be checked for dependencies. Therefore, fetching is part of the intra-group dependency-checking process because it is an integral part of the process.

8. Referring to claim 14, Sites has taught a processor as described in claim 9. Sites has further taught that the intra-group dependency checking is independent of the inter-group dependency checking. It is inherent that these two checks are independent, in a sense, because they are two different checks that happen at two different times (at the very least, they are independent because they happen separately, at different times, and in different pipeline stages). Intra-group checks must be done to see if the instructions can be issued together. This is

dependent on the types of instructions to be issued (some types cannot be issued together). See column 9, lines 46-63. On the other hand, inter-group dependency checking is based on the availability of resources (such as functional units). See column 6, lines 22-29, and note that it is implied that the instructions can issue in parallel, but the resources are not available for one instruction or the other (determined via inter-group checking).

9. Referring to claim 15, Sites has taught a processor as described in claim 9. Sites has further taught that the data dependency and resource allocation checks in earlier pipeline stages of instruction grouping are based, at least in part, on a predicted subsequent state of the superscalar processor. See column 9, lines 50-57, and column 6, lines 42-44, and note that branches may be issued in parallel and branch prediction is utilized. Consequently, all of the dependency and resource checks for instructions in the predicted path of the branch are based on a predicted subsequent state, which is a state subsequent to the state of the system before the prediction occurs.

10. Referring to claim 16, Sites has taught a processor as described in claim 9. Sites has further taught that non-deterministic conditions are evaluated in a final stage of instruction grouping prior to dispatch. See column 9, lines 61-63. *The Free On-line Dictionary of Computing © 1993-2001* defines the term non-deterministic as "Exhibiting nondeterminism," where nondeterminism is defined by the same source as "A property of a computation which may have more than one result." One such condition that is checked would be inter-group dependencies. If an inter-group dependency exists for the two instructions ready to issue, then those instructions will not issue. On the other hand, if an inter-group dependency does not exist for the two instructions, then both of the instructions will be issued simultaneously as a group.

According to the definition above, a dependency will exist or not exist, and consequently, the dependency check has more than one result. Therefore, non-deterministic conditions are evaluated.

11. Referring to claim 17, Sites has taught a processor comprising:

a) plural functional units that execute instructions in respective numbers of processor cycles. See Fig.4 and Fig.6.

b) grouping logic coupled to the functional units and pipelined to compute, over plural cycles, T, of the processor, a future state, S(t + T), of the processor based on a prior state, S(t), of the processor and based thereon to select a group of instructions from a program sequence thereof for dispatch to the functional units, wherein the future state computing takes the plural cycles to complete. See Fig.3 and column 6, lines 22-26, and note that dual issue is possible (issuing a group of two instructions). It should be realized that S(t) is the state in which two instructions are fetched. From stages S0-S3 of the pipeline (Fig.7 and column 10, lines 64-67), it is determined whether these instruction are able to issue together (intra-group checks) and if resources are available for these instructions (inter-group checks). Consequently, in 4 cycles (all of stages S0, S1, S2, and S3, and assuming no stalling), state S(t + T) will finally be computed, where T=4. At this point, the group may be issued to the functional units. Therefore, the future state is calculated over 4 cycles.

12. Referring to claim 18, Sites has taught a processor as described in claim 17. Furthermore, claim 18 is rejected for the same reasons set forth in claim 13.

13. Referring to claim 19, Sites has taught a processor as described in claim 17. Furthermore, claim 19 is rejected for the same reasons set forth in claim 14.

14. Referring to claim 20, Sites has taught a processor as described in claim 17. Sites has further taught:

a) intra-group dependencies are checked by the pipelined grouping logic beginning in a first of the T cycles. See column 10, lines 38-41, and column 9, lines 58-63, and note that if the actual instruction within the group conflict with each other, then they will not be issued together.

b) non-deterministic dependency conditions are checked during a last of the T cycles. See column 9, lines 58-63. *The Free On-line Dictionary of Computing © 1993-2001* defines the term non-deterministic as "Exhibiting nondeterminism," where nondeterminism is defined by the same source as "A property of a computation which may have more than one result." One such condition that is checked would be inter-group dependencies. If an inter-group dependency exists for the two instructions ready to issue, then those instructions will not issue. On the other hand, if an inter-group dependency does not exist for the two instructions, then both of the instructions will be issued simultaneously as a group. According to the definition above, a dependency will exist or not exist, and consequently, the dependency check has more than one result. Therefore, non-deterministic conditions are evaluated. This appears to happen after the intra-group checks because there is no point in determining non-deterministic conditions for both instructions if the instructions cannot issue together anyway. Plus, the passage states that inter-group checks are "also" performed, as in they occur in addition, or after, the intra-group checks.

15. Referring to claim 21, Sites has taught a processor as described in claim 20. Furthermore, claim 21 is rejected for the same reasons set forth in claim 14.

16. Referring to claim 22, Sites has taught a processor as described in claim 17. Sites has further taught that the grouping logic implements T stages of a pipeline of the processor. See

Fig.7 and column 10, lines 64-67, and note that the grouping logic is within the first 4 stages of the pipeline (S0 to S3).

17. Referring to claim 23, Sites has taught a processor as described in claim 17. Sites has further taught that T is four. See Fig.7 and column 10, lines 64-67, and note that the grouping logic is within the first 4 stages of the pipeline (S0 to S3). Note the four stages S0-S3 in the grouping process.

18. Referring to claim 24, Sites has taught a processor as described in claim 17. Sites has further taught that the functional units include at least one functional unit capable of receiving and completing an instruction for each of the processor cycles. See column 9, line 67, to column 10, line 2, and column 2, lines 15-18.

19. Referring to claim 25, Sites has taught a processor as described in claim 17. Sites has further taught that the functional units include at least one functional unit requiring multiple of the processor cycles for receiving and completing an instruction. See column 9, lines 3-5.

20. Referring to claim 26, Sites has taught a method of operating a processor, the method comprising:

a) identifying successive groups of instructions for dispatch to respective ones of plural execution units of the processor. See column 6, lines 18-26. Clearly, successive groups of instructions will continue to be issued to functional units shown in Fig.4 and Fig.6.

b) performing, during plural pipelined execution cycles of the processor, dependency checking amongst instructions of a later one of the groups and between the instructions of the later group and instructions of a preceding one of the groups. In addition, see column 10, lines 38-41, and note that the swap stage of the pipeline (which corresponds to a first cycle) performs intra-group

dependency checking as discussed in column 9, lines 45-63. That is, intra-group dependency checking checks the types of the two fetched instructions and based on the types, determines if they may be issued together. This qualifies as dependency checking because the issuing of instruction A is dependent on instruction B's type (where instructions A and B are the two instructions fetched in any given cycle). Furthermore, see column 10, lines 41-45, and column 6, lines 18-21, and note that after decoding, another dependency check is performed to make sure resources are available. These are inter-group dependency checks because the only reason why resources wouldn't be available for the two instructions is if other instructions are already using those resources (note that this must be the case because resources have not yet been allocated to the fetched instructions). So, this type of dependency check occurs between instructions of different groups. Note also that the intra-group and inter-group checks occur over multiple cycles (stages S0-S3), specifically in stage S1 and again after stage S2 (decode stage) of the pipeline shown in Fig.7. The location of these checks is further illustrated in Fig.3, where it can be seen that the intra-group checks occur at the issue check logic between the instruction register and the decoders and the inter-group checks occur at the issue check logic subsequent to the decoders.

c) dispatching instructions of the later group only after all instructions of the preceding group have been dispatched. See column 6, lines 26-38 and note that a remaining instruction from a group will be issued before an instruction of another group.

d) wherein the performed dependency checking takes the plural pipelined execution cycles to complete. Again, note that the intra-group and inter-group checks occur over multiple cycles (stages S0-S3), specifically in stage S1 and again after stage S2 (decode stage) of the pipeline

shown in Fig.7. The location of these checks is further illustrated in Fig.3, where it can be seen that the intra-group checks occur at the issue check logic between the instruction register and the decoders and the inter-group checks occur at the issue check logic subsequent to the decoders.

21. Referring to claim 27, Sites has taught a method of operating a processor as described in claim 26. Sites has further taught that the dependency checking is performed by pipelined grouping logic. From stages S0-S3 of the pipeline (Fig.7 and column 10, lines 64-67), instructions are fetched and it is determined whether these instructions are able to issue together based on instruction types and resources used by these instructions (intra-group checks) and if resources are available for these instructions (inter-group checks). Therefore, stages S0-S3 of the pipeline make up the pipelined grouping logic.

22. Referring to claim 28, Sites has taught a method of operating a processor as described in claim 26. Sites has further taught that intra-group dependency checking and within-group resource allocation are performed in successive processor cycles by pipelined grouping logic. See column 6, lines 26-29, and note that even if the instructions may issue together (passes intra-group checks), they still may be unable to issue together if resources are only available for one instruction (determined by inter-group checks). If this is the case, resources are only allocated to the one instruction of the two. If resources are available for both instructions in the group, then both may issue at the same time. Therefore, resource allocation happens only when the instructions are issued and to be issued, the intra-group checks must first be made.

23. Referring to claim 29, Sites has taught a processor as described in claim 26. Furthermore, the method of claim 29 is performed by the processor of claim 14. Consequently, claim 29 is rejected for the same reasons set forth in claim 14.

24. Referring to claim 30, Sites has taught a method of operating a processor as described in claim 26. Sites has further taught that non-deterministic conditions are evaluated in a final one of the pipelined execution cycles implemented by pipelined grouping logic. See column 9, lines 58-63. *The Free On-line Dictionary of Computing © 1993-2001* defines the term non-deterministic as "Exhibiting nondeterminism," where nondeterminism is defined by the same source as "A property of a computation which may have more than one result." One such condition that is checked would be inter-group dependencies. If an inter-group dependency exists for the two instructions ready to issue, then those instructions will not issue. On the other hand, if an inter-group dependency does not exist for the two instructions, then both of the instructions will be issued simultaneously as a group. According to the definition above, a dependency will exist or not exist, and consequently, the dependency check has more than one result. Therefore, non-deterministic conditions are evaluated. This appears to happen after the intra-group checks because there is no point in determining non-deterministic conditions for both instructions if the instructions cannot issue together anyway. Plus, the passage states that inter-group checks are "also" performed, as in they occur in addition, or after, the intra-group checks.

25. Referring to claim 31, Sites has taught a method of grouping instructions for dispatch to execution units of a processor, the method comprising:

a) in a first cycle of processor execution, identifying plural candidate instructions for an instruction group. See Fig.3 and note that two instructions are fetched (stage S0 of the pipeline shown in Fig.7) and identified as instructions that may be issued together.

b) in a subsequent cycle of processor execution, beginning intra-group dependency checking as amongst instructions of the instruction group. See column 10, lines 38-41, and column 9, lines

50-57, and note that that instructions must be of the types shown in the table, otherwise a conflict exists between the two instructions and they cannot issue together. Also, the checks will occur after stage S0 in the pipeline (since S0 is the fetch stage) in the S1 stage. More specifically, the instructions must be fetched before they're checked for intra-group dependencies.

c) in a cycle of processor execution prior to dispatch of any instruction from the instruction group, checking non-deterministic conditions. See column 6, lines 18-29. *The Free On-line Dictionary of Computing © 1993-2001* defines the term non-deterministic as "Exhibiting nondeterminism," where nondeterminism is defined by the same source as "A property of a computation which may have more than one result." One such condition that is checked would be inter-group dependencies. If an inter-group dependency exists for the two instructions ready to issue, then those instructions will not issue. On the other hand, if an inter-group dependency does not exist for the two instructions, then both of the instructions will be issued simultaneously as a group. According to the definition above, a dependency will exist or not exist, and consequently, the dependency check has more than one result. Therefore, non-deterministic conditions are evaluated. The inter-group checking occurs after the decode stage (S2) and prior to issue (since the checks help decide when to issue what instructions).

d) in a cycle of processor execution prior to the non-deterministic dependency condition checking, initiating inter-group dependency checking as between instructions of the instruction group and instructions of one or more prior instruction groups. Clearly, before the non-deterministic dependency checking occurs (which is the inter-group checking), the inter-group checking must be initiated. And, it is clear that inter-group checking occurs between groups and that this type of check exists within Sites. That is, if an instruction from group "A" is already

using a floating-point divide unit, for instance, then the successive instruction from group "B" will have to wait until that unit is available.

e) wherein, for instruction instances of the instruction group, dependency checking, which includes the intra-group dependency checking and the inter-group dependency checking, takes plural of the cycles of processor execution to complete. From stages S0-S3 of the pipeline (Fig.7 and column 10, lines 35-47, and lines 64-67), instructions are fetched and it is determined whether these instructions are able to issue together based on instruction types and resources used by these instructions (intra-group checks) and if resources are available for these instructions (inter-group checks). Therefore, the intra-group and inter-group checks occur over multiple cycles (stages S0-S3), specifically in stage S1 and again after stage S2 (decode stage) of the pipeline shown in Fig.7. The location of these checks is further illustrated in Fig.3, where it can be seen that the intra-group checks occur at the issue check logic between the instruction register and the decoders and the inter-group checks occur at the issue check logic subsequent to the decoders.

26. Referring to claim 32, Sites has taught a method as described in claim 31. Furthermore, the method of claim 32 is performed by the processor of claim 12. Therefore, claim 32 is rejected for the same reasons set forth in claim 12.

27. Referring to claim 33, Sites has taught a method as described in claim 31. Sites has further taught that the non-deterministic condition checking is performed conservatively, based on an assumption of no change in condition. Clearly, it must be assumed that that there is no change in condition, i.e., dependencies exist (recall that non-deterministic condition checking is

synonymous with the inter-group checking in that when performing inter-group checks, a dependency either exists or does not exist). Consequently, dependencies must be checked for.

28. Referring to claim 34, Sites has taught a method as described in claim 31. Sites has further taught:

a) that the non-deterministic condition checking is performed aggressively, computing dispatch conditions for at least two alternatives including no change in condition and condition resolution. Again, the non-deterministic condition checking is synonymous with the inter-group checking in that when performing inter-group checks, a dependency either exists (no change in condition) or does not exist (condition resolution).

b) a control signal is selective for a particular one of the computed alternatives. Clearly, different things happen when a dependency exists (instructions not issued) as opposed to when doesn't exist (instructions are issued). In order to determine which action to take, a control signal is inherently present.

(10) Response to Argument

Regarding claims 26, 27, 29, and 30

On page 5 of the brief, appellant's summary of argument (1) is listed. This argument is explained further on pages 7-9 of the brief with respect to claim 26. On page 7, last paragraph, a minor error was noted in appellant's argument. The phrase "(S0, S1, or S3)" should in fact be "(S0, S1, or S2)".

Appellant argues in the last paragraph on page 7 of the brief, that:

> "Contrary to the Office's position, dependency checking in Sites is not performed over plural pipelined execution cycles, but rather, is performed conventionally, during a single cycle, just prior to issue, in the fourth stage (S3) of Sites' pipeline. In Sites, four pipeline stages are employed prior to issue but, it is incorrect to interpret any of the first three stages (S0, S1, or S2) as "dependency checking." Instead, a first stage (S0) is the fetch stage, a second stage (S1) is the swap stage, a third stage (S2) is the decode stage, and a fourth stage (S3) is the register file access and issue stage. Sites, col. 10, lines 26-47."

These arguments are not found persuasive for the following reasons:

a) The first four stages (S0-S3) of Sites' pipeline (Fig.7) are the focus of discussion. As appellant stated, stage S0 is a fetch stage in which two instructions are fetched from cache memory (column 10, lines 35-38). Stage S1 is a swap stage, during which the two fetched instructions are evaluated to see if they can be issued at the same time (column 10, lines 38-41). Stage S2 is the decode stage, during which the two instructions are decoded (column 10, lines 41-44). And, stage S3 is the register file access and issue stage in which an issue check is made for all instructions (column 10, lines 45-47).

The crux of the argument appears to be that "dependency checking" is only performed in a single cycle/stage (S3) and not performed over multiple cycles/stages, as the examiner contends, because it is incorrect to interpret any of the first three stages (S0, S1, or S2) as performing "dependency checking."

The examiner respectfully disagrees and asserts that in the swap stage (S1), two instructions are evaluated to see if they can be issued at the same time (column 10, lines 38-41). More specifically, in Sites, only certain types of instructions may be paired together. For whatever reason (whether it be resource constraint or some other constraint), Sites does not allow certain types of instructions to be paired (column 9, lines 45-63). Note that an example is given where an integer load/store instruction may be paired with an integer operate instruction, but an integer load/store instruction may not be paired with an integer branch instruction. From this

example, it can be seen that in one cycle (in the S1 stage), the types of the two fetched instructions are checked. If the types conflict, then dual issue cannot occur. If the types are compatible, then dual issue can occur (i.e., the instructions are paired). The examiner contends that this stage performs intra-group dependency checking. This checking qualifies as "intra-group" checking because the check occurs among instructions in the fetched group. And, this checking also qualifies as dependency checking because the issuing of an instruction in the group is dependent on the other instruction's type. That is, if instructions A and B are fetched in stage S0, then in stage S1, the decision to issue instruction B is dependent on instruction A's type.

Furthermore, a second type of dependency checking occurs in another cycle, i.e., after stage S2 of the pipeline. More specifically, once it is determined that the two fetched instructions may issue together (based on the intra-group dependency checks in a previous cycle (stage S1)), the instructions are sent to the decoder to be decoded (column 10, lines 41-44). Then, after the instructions are decoded, the system makes another check to see if resources are available (column 6, lines 18-21). If resources are only available for one of the two instructions, then only that one instruction will issue. If resources are available for both, and dual issue was determined to be possible in stage S1 as a result of the intra-group dependency check, then both instructions will issue (column 6, lines 21-29). The examiner asserts that these checks occurring after decoding are inter-group dependency checks. That is, once dependencies are checked among instructions in a first group, checks must then occur between the first group and a previous group (i.e., a group that has already issued). Clearly, this second type of checking occurs between groups because column 6, lines 18-21, disclose checking to see if resources are available for the two fetched instructions. Since the resources have not yet been allocated to the

two fetched instructions, the only reason they wouldn't be available is if other instructions are using them. Hence, the inter-group checking is for checking if another group is using resources that the current group needs. This type of checking is essential for at least the following reasons:

a) to resolve RAW hazards. A RAW hazard occurs when a subsequent instruction reads a register that has not yet been written to by a previous instruction. For instance, if the first instruction to execute were an "ADD R0, R1, R2" instruction (add the contents of registers R1 and R2 and store the result in R0) and the second and subsequent instruction is an "ADD R3, R4, R0" instruction (add the contents of registers R4 and R0 and store the result in R3), then if the first instruction does not complete before the second instruction reads R0, the second instruction will not be using the correct value in R0. As a result, a dependency check needs to be made so that the second instruction is held back from issuing until the first instruction writes to R0. It should be noted that RAW hazards must be resolved; otherwise, the execution process may generate invalid results.

b) to resolve structural hazards. A structural hazard occurs when there are not enough resources to satisfy all requesting instructions. For instance, if there is one floating-point unit which executes one floating-point instruction every X cycles, then there is no way that the unit can execute multiple instructions in those X cycles. Essentially, one execution unit may execute one instruction, two execution units may execute two instructions, but two execution units may not execute three instructions. Therefore, if an instruction is already using a unit, and an instruction in a subsequent group also needs that unit, it must wait until the first instruction is done. This is another type of hazard which must be resolved.

Appellant appears to concede that this inter-group dependency checking does occur (last paragraph on page 7 of the brief), and so the main argument is that dependency checking does not occur in a cycle previous to the S3 cycle. For the reasons above, the examiner believes that intra-group checking is performed in stage S1 of the pipeline, which is before decoding (stage S2), and consequently, before the inter-group checking (which appellant appears to concede). This is further supported by merely looking at Fig.3 of Sites in which a first issue check (intra-group check) is performed before decoding and a second issue check (inter-group check) is performed after decoding.

Appellant argues on pages 8-9 of the brief, that:

> (1) "...the relied upon disclosure does not concern "dependency checking." Second, even with respect to resource availability, the Office's reliance is inconsistent with the full passage. Indeed, the more complete quote from Sites drives home the point that Sites' check for availability of required resources is performed in the stage immediately prior to supply of instructions and register, i.e., after decode (stage S2) in issue stage S3." (brief, 1st half of page 8)
>
> (2) "The Office provides no real factual basis for it's next supposition that "dependencies and resources are checked over multiple cycles (stages S0-S3)... Since information regarding operations and register addresses would necessarily be employed in any dependency checking in Sites to identify the source and target conflicts between instruction, it follows that dependency checking in Sites must be performed, if at all, in stage S3 after such register addresses and operations are decoded." (brief, middle of page 8 to page 9)

These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, the examiner does not disagree with appellant's argument that checking occurs after decode. In fact, the examiner is relying on such checking as being the inter-group dependency checking that is claimed throughout the application.

b) Regarding the second argument, again, the examiner does not dispute that inter-group dependency checking occurs after decoding. However, appellant appears to be interpreting "dependency checking" too narrowly. Appellant appears to be arguing that dependency

checking necessarily involves operations and register addresses determined after decoding (note the underlined portion of the reproduced argument above). However, appellant's specification hardly requires that dependency checking be defined this way. It should be noted that the examiner did not evaluate appellant's claims in a vacuum. The specification was consulted and no explicit definition was found for "dependency checking". The checking in the swap stage (S1) of Sites qualifies as dependency checking because, as previously stated, a determination is made to see whether instruction B may be dual-issued with instruction A, and instruction B's issuance is dependent on instruction A's type. The examiner feels that this is a perfectly valid interpretation of one type of dependency checking. And, since this qualifies as dependency checking (intra-group checking) along with the checking that occurs after decoding (inter-group checking), it is concluded that multiple dependency checks are made over multiple cycles in Sites (before and after decoding).

Appellant argues on page 9 of the brief, that:

> "...Regarding stalls, it is unclear from the Advisory Action whether the Office continues to rely on a stall cycle argument; however, assuming (for the sake of completeness) that the Office continues to rely on such reasoning, Applicant respectfully submits that an operation that is performed in one cycle is still performed in one cycle whether or not, after performance of that operation, the pipeline is stalled for one or more subsequent cycles."

These arguments are not found persuasive for the following reasons:

a) As explicitly stated in the Advisory Action mailed on February 17, 2005, the stalls are not being relied upon to anticipate appellant's claims. For the reasons above, which don't include the idea of stalling, the examiner believes Sites reads on appellant's claims. Stalling was pointed out in Sites as an example, but the examiner believes that this point is overall irrelevant in light of the reasoning above, which has been maintained over time by the examiner.

Regarding claim 28

Appellant argues on pages 9-11 of the brief, that:

> The Office apparently reasons "that a single instruction (of pair) issuance scenario might result in performance (again, apparently in a 2nd pass) of within-group resource allocation for the remaining un-issued instruction of the pair."
>
> "Applicant respectfully notes that Sites does not disclose this hypothecated 2nd pass within-group resource allocation. Indeed, other than indicating that the instruction unit stalls,... Sites is silent on pipeline behavior in a single instruction (of pair) issuance scenario. Furthermore, a 2nd pass within-group resource allocation operation does not follow from that which Sites does disclose. More importantly, such a mode of operation does not really make sense in the Sites dual-issue architecture since once a single instruction (of the pair) issues, only one instruction remains and no role for "within-group [of one] resource allocation" is apparent. Indeed, the Office's interpretation (in effect) construes the claim limitation as a nullity, since there is no "within-group resource allocation" to be performed amongst members of a group that consists of a single remaining valid instruction available to be issued."
>
> "In the present case, the hypothecated successive-cycle performance of intra-group dependency checking and within-group resource allocation is not necessarily present in Sites and cannot be said to necessarily follow from Sites' disclosure. Indeed, it is not really plausible (or probable) that, after issuing one (of two) instructions, the processor disclosed by Sites would necessarily repeat "within-group resource allocation" for a "group" that includes but one unissued instruction.

These arguments are not found persuasive for the following reasons:

a) For each of the arguments above, it is not exactly clear what appellant is arguing. Appellant appears to be arguing limitations which are not in the claim (2nd pass allocation). All the claim requires is that intra-group checking and within-group resource allocation occur in successive cycles. As described above, the intra-group checking occurs in stage S1 of the pipeline. Resource allocation for instructions in the group succeeds this checking because resource allocation occurs after decoding which occurs after the intra-group checks. See column 6, lines 18-29. The examiner, in the rejection of claim 28, has pointed out that this is the case by stating "Therefore, resource allocation happens only when the instructions are issued and to be issued, the intra-group checks must first be made." That is, once intra-group checking finishes, and the

instructions may be issued together, resource allocation will occur for at least one instruction in the group. In the rejection, the examiner also pointed out an example explained by Sites in the aforementioned passage. Specifically, resources may be allocated to one instruction in the group if that is all that is available. On the other hand, resources will be allocated to both instructions in the group if they are available.

Regarding claim 31

Appellant argues on page 11 of the brief, that:

> "In rejecting the claims, the Office conveniently reads the word "dependency" out of the claim. The Office apparently does so to avoid the problem that , in Sites design, "intra-group dependency checking" and "inter-group dependency checking" must be performed, if at all, in one and only one pipeline stage, namely stage S3, after register addresses are decoded, to even allow an evaluation of whether a dependency exists."

These arguments are not found persuasive for the following reasons:

a) The examiner completely disagrees that the word "dependency" was conveniently read out of the claim. For the reasons above with respect to claim 26, it is clear why the examiner believes that the checking at stage S1 of Sites and again at a stage after decoding in Sites constitutes dependency checking. Just because appellant's argued definition of "dependency checking" does not match the examiner's interpretation of the claimed dependency checking (which was not explicitly defined in the consulted specification), it does not mean that limitations were ignored.

Appellant argues on the first half of page 12 of the brief, that:

> (1) "...whatever check may even arguably be hinted at for swap stage S1, it is not dependency check but rather a resource check."
>
> (2) "Finally, it is notable that the Office asserts elsewhere in its rejection of claim 31 (see

> Final Action, paragraph 34(b)) that an "intra-group check" in Sites is performed "after instructions are decoded" (i.e., in stage S3 following decode stage S2 and consistent with the interpretation urged by Applicants). The Office cannot construe "intra-group dependency check" one way (i.e., as a stage S3 operation) to dispose of a first limitation, then another inconsistent way (i.e., as a stage S0-S2 operation) to dispose of another limitation of the same claim. For this reason as well, claim 31 and those dependent therefrom (claims 32-34) are allowable over Sites."

These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, it is not clear how the checking of stage S1 is not a dependency check. Stage S1 checks the types of two fetched instructions and, based on the type, determines if dual issue is possible. That is, if instructions A and B are fetched in stage S0, then in stage S1, a dependency check is made because the decision to issue instruction B is dependent on instruction A's type. This is a form of dependency checking because one instruction's fate is dependent on the type of the other instruction. As a side note, appellant appears to be arguing that dependency checking is not equivalent to resource checking ("...it is not a dependency check but rather a resource check,"). It is not clear to the examiner how the check of stage S1 may be considered a resource check but not considered a dependency check, especially given the aforementioned reasoning.

b) Regarding the second argument, the examiner asserts that the rejection of claim 31 (part b) included a minor error. That is, the examiner described the intra-group checks (by referring to column 9, lines 50-57) but additionally, and mistakenly, pointed to a passage describing the inter-group checks (column 6, lines 18-21). The examiner feels that it was made clear throughout the action which checks are intra-group checks and which checks are inter-group checks. The teaching of intra-group checks was still pointed out in the rejection, but a passage was incorrectly cited. This passage has been removed from the rejection for clarity.

Appellant argues in a footnote on pages 11-12 of the brief, that:

> "Applicant specifically reserves the issue of whether, properly construed, Sites should be understood to enablingly disclose any intra-group check at swap stage S1, nothing that while the Examiner chooses to rely on a portion of the statement: "The second stage S1 is the swap stage, during which the fetched instructions are evaluated by the circuit 25 to see if they can be issued at the same time (Sites, col. 10, lines 38-41)". The weight of the description as a whole establishes that dual issue decision making is performed by circuit 25, which necessarily places such performance in stage S3, since circuit 25 follows decode logic (and associated stage S2) in the pipeline flow. In short, the Office has ignored the clear statement(s) attributing dual issue decision making to circuit 25 (stage S3) in favor of one inconsistent (and arguably errant) statement that better supports its theory of anticipation."

These arguments are not found persuasive for the following reasons:

a) The examiner feels that it was clearly expressed (and illustrated) by Sites in column 10, lines 38-41, that a check occurs in stage S1, which is before the decode stage. Although circuitry 25 (of Fig.3) is disclosed as performing such a check, and circuitry 25 is labeled as being after the decoder, one should not ignore the explicit issue check that occurs before decoding. That is, by looking at Fig.3, after instructions are fetched into the instruction register 22, and before the instructions are decoded, a first check is performed by first issue-check logic. The first issue check logic does not have a reference number associated with it. It may very well be a part of circuitry 25. One thing is certain from Fig.3 and the cited passages of Sites ,and that is that there are two issue checks: one before decoding and one after decoding. These checks therefore occur over multiple cycles.

Regarding claim 34

Appellant argues on pages 6-7 (argument (5)), and on page 12 of the brief, that:

> "...Applicant's design computes outcomes for both of at least two alternative possibilities of a non-deterministic condition whose result cannot be precisely determined based on information available at the time In this way, on either result, a dispatch condition has been computed. There is simply no disclosure or suggestion of such an approach in Sites."

These arguments are not found persuasive for the following reasons:

a) Regarding "non-deterministic," the examiner consulted the specification to determine what was meant by such a term (i.e., the claims were not evaluated in a vacuum). There is no explicit definition of non-deterministic conditions. Instead, the following statements were made in appellant's specification:

1) "…most components of next state S(t+1) can be precisely determined from present state S(t), and the remaining components of state S(t) can be reasonably determined, provided that certain non-deterministic conditions are appropriately handled." (page 8, lines 11-15)

2) "Finally, at stage 204, non-deterministic conditions, e.g., the condition at store buffer 105, is considered." (page 11, lines 29-31).

As a result, appellant has not explicitly defined such a condition, but rather gives one example of it. The examiner therefore found a definition of non-deterministic in *The Free On-line Dictionary of Computing © 1993-2001,* which defines the term "non-deterministic" as "Exhibiting nondeterminism," where "nondeterminism" is defined by the same source as "A property of a computation which may have more than one result." One such condition that meets this description would be inter-group dependencies. That is, computing inter-group dependencies for a given input results in either a dependency existing or not existing for the two instructions ready to issue. If a dependency does exist, then those instructions will not issue. On the other hand, if an inter-group dependency does not exist for the two instructions, then both of the instructions will be issued simultaneously as a group. According to the definition above, a dependency will exist or not exist, and consequently, the dependency check has more than one result. Therefore, non-deterministic conditions are evaluated. If appellant wanted non-

deterministic conditions to be related to a store buffer, as argued, then this limitation should have been claimed.

Regarding claims 9-12 and 14

Appellant argues on pages 13-14 in a manner similar to the arguments of claim 26. That is, it is asserted by appellant that Sites has not taught intra-group and inter-group dependency checking occurring over plural execution cycles. These arguments are not found persuasive for the same reasons given in response to the arguments of claim 26 above. In short, intra-group dependency checks occur in stage S1 of Sites' pipeline and inter-group dependency checks occur after decoding, in stage S3 of the pipeline.

Regarding claim 13

Appellant argues on page 6 (argument (3)) and on page 14 of the brief, that:

> "As a preliminary matter, applicant notes that Sites swap stage (S1) does not constitute intra-group dependency checking. However, even assuming arguendo the Office's faulty premise, it is clear and unambiguous that the preceding stage (fetch stage S0) has nothing at all to do with dependency checking...Nonetheless, the Office construes Sites' first two pipeline stages, i.e., S0 and S1, as intra-group dependency checking that spans multiple cycles."

These arguments are not found persuasive for the following reasons:

a) It should be noted from claims 9 and 13 that intra-group dependency checking is not defined in terms of functionality. And, as discussed above, there is no explicit definition of intra-group dependency checking in the instant specification. Consequently, the examiner asserts that fetch stage S0 and swap stage S1 form the overall intra-group dependency check process. That is, before two instructions are checked for dependencies, they must be fetched. Therefore, fetching is an integral part of intra-group dependency checking. Without fetching, intra-group checks

could not be performed because there would be no group to check. Again, it should be realized that appellant has not defined what constitutes a dependency check. As far as the claims are concerned, why could intra-group dependency checking not include fetching, especially since it is an integral part? Based on the examiner's rejection, because the intra-group checking does comprise stages S0 and S1, it is clear that the intra-group checking alone spans multiple execution cycles (cycles S0 and S1), which is what is recited in claim 13.

Regarding claim 15

Appellant argues on page 15 of the brief, that:

> "In rejecting claim 15, the Office relies on the disclosure of Sites related to branch prediction, not predicted subsequent state. In particular, while Sites processor may actually execute instructions based on predicted taken (or untaken) branch, any data dependency checking and/or resource allocation checks performed are based on an actual, the-current state of Sites' processor."

These arguments are not found persuasive for the following reasons:

a) Suppose at time T, a branch instruction is encountered in the system of Sites. Since branch prediction is employed by Sites (column 6, lines 42-48), the branch is predicted. For sake of argument, let's assume the branch is predicted taken and the taken branch path is followed. The predicted subsequent state with respect to time T is the predicted-taken state. Now, any dependency checks that occur until that branch is resolved (at time T+X) are based on the predicted-taken state, which is a predicted, subsequent state with respect to time T. Yes, at any given time, the system is in a current state (so to speak), but the fetching and checking of the instructions along a predicted path are based on the predicted state. If the prediction turns out to be incorrect, then the dependency checks were basically for naught because the instructions for

which the checks were done were not supposed to execute anyway. For this reason, it can be seen that the checks are based on a predicted state.

Regarding claims 17, 19, and 22-25

On page 15 of the brief, in the first sentence under the heading "Claims 17, 19, and 22-25, a minor error was detected. The phrase "With regard to claim 18" should in fact be "With regard to claim 17".

Appellant argues on pages 15-16 of the brief, that:

> "In this regard, the Office apparently misinterprets Sites initial pipeline stages. In Sites, three stages (fetch S0, swap S1 and decode S2) lead up to a fourth stage (register access/issue S3) in which an issue decision (e.g., a future state S(t+1)) is made (calculated in a single cycle) based on the processor state that then exists (e.g., the then current state S(t)). Operations performed in preceding stages S0, S1 and S2 cannot be properly interpreted as computing a future state of Sites processor based on then current state. In particular,
>
> 1. a fetch performed at stage S0 does not compute a future state of the processor (4-cycles ahead) based on state of the processor in the then-current cycle;
>
> 2. a swap performed at stage S1 does not compute a future state of the processor 3-cycles ahead) based on state of the processor in the then-current cycle; and
>
> 3. a decode performed at stage S2 does not compute a future state of the processor (2-cycles ahead) based on state of the processor in the then-current cycle.
>
> Instead, all future state computing in Sites is performed (1-cycle ahead) in a single-cycle register access/issue stage S3 based on state of the processor in the then-current cycle. Accordingly, Sites does not disclose or suggest the invention recited in claim 17.
>
> Perhaps most instructive in diagnosing the Office's faulty reasoning is its conclusion: "Consequently, in 4 cycles (stages S0 -S3 and assuming no stalling), state S(t+T) will finally be computed."). It is not sufficient that a future state will finally be computed by single (fourth) stage of the pipeline, rather the claim requires grouping logic pipelined to compute over plural cycles, T, a future state, S(t+T), of the processor based on a prior state, S(t), of the processor.
>
> Sites does not disclose or suggest the required computation over plural cycles of a future state, S(t+T), of the processor based on a prior state, S(t). Accordingly, claim 17 and those dependent therefrom (claims 18-25) are all allowable."

These arguments are not found persuasive for the following reasons:

a) The examiner asserts that appellant is reading the claim too narrowly. If you take the claim language "...compute, over plural cycles, T, of the processor, a future state, S(t+T), of the processor based on a prior state, S(t)," it should be realized that Sites teaches exactly this. More specifically, state S(t) in Sites is the state in which two instructions are fetched (beginning of stage S0). Based on the two instructions fetched (based on the initial state), a future state S(t+T), with respect to the initial state, is calculated. That is, over time (over a span of 4 pipeline cycles (all of stages S0-S3)), a future state S(t+T) is calculated with respect to the prior state. What happens in the stages after S0 is dependent on what happens in S0, and therefore, the future state S(t+T) is dependent on state S(t).

There is no language in the claims which explains what a future state is. In addition, a future state is not finally computed by a single fourth stage of the pipeline, as appellant suggests. Instead, the state is computed over plural pipeline cycles. If stage S0 corresponds to state S(t), then at the end of S0 (one cycle has elapsed), S(t+1) is computed which corresponds to stage S1. At the end of S1 (another cycle has elapsed), S(t+2) is computed which corresponds to stage S2. At the end of S2 (another cycle has elapsed), S(t+3) is computed which corresponds to stage S3. And, at the end of S3 (another cycle has elapsed), S(t+4) is computed. So, S(t+4) is computed over a span of 4 pipeline cycles, where T=4. S(t+4) corresponds to the future state of S(t).

Regarding claim 18

Appellant argues on pages 16-17 in a manner similar to the arguments of claim 13. Even though the scope of claim 18 is admittedly different from that of claim 13, the examiner response to the arguments for claim 18 are the same as the response to the arguments of claim 13. That is, the

fetch stage and swap stage make up the intra-group checking process, which would take multiple cycles.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

DJH
David J. Huisman
November 7, 2006

DONALD SPARKS
SUPERVISORY PATENT EXAMINER

Conferees:

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100